“Fixing” the specification of widenings

Enea Zaffanella and Vincenzo Arceri

Abstract The development of parametric analysis tools based on Abstract Interpretation relies on a clean separation between a generic fixpoint approximation engine and its main parameter, the abstract domain: a safe integration requires that the engine uses each domain operator according to its specification. Widening operators are special, among other reasons, in that they lack a single, universally adopted specification. In this paper we review the specification and usage of widenings in several open-source implementations of abstract domains and analysis tools. While doing this, we witness a mismatch that potentially affects the correctness of the analysis, thereby suggesting that the specification of widenings should be “fixed”. We also provide some evidence that a fixed widening specification matching the classical one allows for precision and efficiency improvements.

1 Introduction

Abstract Interpretation (AI) [17, 18, 19] is a mature research field, with many years of research and development that led to solid theoretical results and strong practical results. Among the many maturity indicators, the most important ones are probably the following:

- the development of AI tools (analyzers, verifiers, etc.) is firmly based on the theoretical results; while some of these tools happen to take shortcuts for practical reasons, these are usually identifiable and quite often well documented;

Enea Zaffanella
University of Parma, Parco Area delle Scienze 53/A, 43124, Parma, Italy e-mail: enea.zaffanella@unipr.it

Vincenzo Arceri
University of Parma, Parco Area delle Scienze 53/A, 43124, Parma, Italy e-mail: vincenzo.arceri@unipr.it

- some AI tools have been industrialized and commercialized; we only mention here the remarkable case of Astrée [10];
- there is growing support for collaborative work, including open-source libraries of abstract domains and AI frameworks; hence, it is relatively easy to extend, combine and compare AI tools;
- in recent years, the synergy between theory and practice has been fostered by the rather common usage of pairing research papers with corresponding artifacts, allowing for repeatability evaluations and even competitions between AI tools.

As a rough attempt at summarizing all the positive consequences of working in a mature research field, one may state the following: the knowledgeable members of the AI community (which are not necessarily experts) know what it is needed in order to have things go right; also, *they can tell when things are going wrong*.

In this paper we will question the truth of the last part of the sentence above: we will provide some evidence that, in the recent past, things may have gone wrong without people noticing (even the experts).

We start by informally introducing the problem. The implementation of an AI-based static analysis tool is greatly simplified by pursuing a clean separation of its different components [35]: in particular, the development of a generic *AI engine* (i.e., the fixpoint approximation component) is kept distinct from the development of its main parameter, the *abstract domain*. Quite often, these two components are designed and implemented by different people, possibly working for distinct organizations. A safe integration requires that the AI engine implements its functionalities by using a subset of the semantic operators provided by the abstract domain and, in particular, that each of these operators is used according to its formal specification. In order to enable and foster collaborative work, it is therefore essential that the AI research community converges towards a well-defined, uniform interface for these operators. While this goal can be considered achieved for most semantic operators, it has been surprisingly missed for *widening operators*: as we will see, widenings happen to be implemented and used according to slightly different and incompatible specifications, thereby becoming a potential source of interface mismatches that can affect the correctness of AI tools. In this paper we will review the specification and usage of widenings in several open-source implementations of abstract domains and analysis tools, witnessing the above mentioned mismatch. We will also propose a solution that, in our humble opinion, represents a further little step towards the maturity of the research field: namely, we suggest to "fix" the specification of widenings by choosing, once and for all, the one that more likely allows for preserving, besides correctness, also efficiency and precision.

The paper is organized as follows. In Section 2 we briefly recall AI-based static analysis and the classical specification for widening operators. In Section 3, after introducing an alternative specification for widenings, we classify open-source abstract domain and AI engine implementations according to the adopted widening specification. Building on this classification, in Section 4 we describe and compare the possible kinds of abstract domain and AI engine combinations, discussing the safety problems potentially caused by specification mismatches. In Section 5 we propose to fix the specification of widenings, so as to solve the identified problems

and also simplify the development of correct abstract domains and AI engines, while avoiding inefficiencies and precision losses. We conclude in Section 6.

2 Background

In this section we briefly recall some basic concepts of Abstract Interpretation (AI) [16, 17, 18, 19, 20]. Interested readers are also referred to [41], which provides an excellent tutorial focused on numerical properties. For exposition purposes, we will describe here the classical AI framework based on Galois connections; however, as discussed at length in [19], it is not difficult to preserve the overall correctness of the analysis even when adopting weaker algebraic properties.

The semantics of a program is specified as the least fixpoint of a continuous operator $f_C \colon C \to C$ defined on the concrete domain C, often formalized as a complete lattice $\langle C, \sqsubseteq_C, \sqcup_C, \sqcap_C, \bot_C, \top_C \rangle$. In most cases, the partial order relation $\sqsubseteq_C$ models both the *computational ordering* (used in the fixpoint computation) and the *approximation ordering* (establishing the relative precision of the concrete properties). The least fixpoint can be obtained as the limit

$$\mathrm{lfp} f_C = \sqcup_C \{\, c_i \mid i \in \mathbb{N} \,\}$$

of the increasing chain $c_0 \sqsubseteq_C \cdots \sqsubseteq_C c_{i+1} \sqsubseteq_C \cdots$ of Kleene's iterates, defined by $c_0 \stackrel{\text{def}}{=} \bot_C$ and $c_{i+1} \stackrel{\text{def}}{=} f_C(c_i)$, for $i \in \mathbb{N}$.

The goal of AI-based static analysis is to soundly approximate the concrete semantics using a simpler abstract domain A, which is usually (but not always) formalized as a bounded join semi-lattice $\langle A, \sqsubseteq_A, \sqcup_A, \bot_A, \top_A \rangle$. Intuitively, the relative precision of the abstract elements is encoded by the abstract partial order $\sqsubseteq_A$, which should mirror the concrete one. Formally, the concrete and abstract domains are related by a *Galois connection*, which is a pair of monotone functions $\alpha \colon C \to A$ and $\gamma \colon A \to C$ satisfying

$$\forall c \in C, \forall a \in A : \alpha(c) \sqsubseteq_A a \Leftrightarrow c \sqsubseteq_C \gamma(a).$$

When C and A are related by a Galois connection, an abstract function $f_A \colon A \to A$ is a *correct approximation* of the concrete function $f_C \colon C \to C$ if and only if $\alpha(f_C(c)) \sqsubseteq_A f_A(\alpha(c))$ for all $c \in C$ or, equivalently, if $f_C(\gamma(a)) \sqsubseteq_C \gamma(f_A(a))$ for all $a \in A$. Note that correct approximations are preserved by function composition; hence the problem of approximating the concrete function f_C is usually simplified by decomposing it into simpler functions (depending on the semantics of the considered programming language), which are then approximated individually. As a matter of fact, software libraries implementing abstract domains usually provide several abstract operators approximating the most common concrete semantic operators (addition/removal of program variables, assignments, conditional tests, merging of different control flow paths, etc.).

By exploiting the correctness condition above, traditional AI-based static analysis tools approximate the concrete semantics by computing *abstract Kleene iterates*. In

principle, they will compute the sequence $a_0, \ldots, a_{i+1}, \ldots$ defined by $a_0 \stackrel{\text{def}}{=} \bot_A$ and $a_{i+1} \stackrel{\text{def}}{=} f_A(a_i)$, for $i \in \mathbb{N}$: by induction, each abstract iterate is a correct approximation of the corresponding concrete iterate, i.e., $c_i \sqsubseteq_C \gamma(a_i)$; moreover, if the abstract sequence converges to an abstract element $a \stackrel{\text{def}}{=} \sqcup_A \{ a_i \mid i \in \mathbb{N} \} \in A$, then the correctness relation also holds for the least fixpoint, i.e., $\mathrm{lfp} f_C \sqsubseteq_C \gamma(a)$.

In general, however, the abstract sequence may not converge at all (since the abstract function f_A is not required to be monotone nor extensive and neither the abstract domain is required to be a complete lattice) or it may fail to converge in a finite number of steps. The classical method to obtain a finite convergence guarantee is by the use of *widening operators*; we recall here the classical specification for widening operator as provided in [16, 17, 20] (see also [41]).

Definition 1 (Classical widening)

A widening $\triangledown \colon A \times A \to A$ is an operator such that:

1. $\forall a_1, a_2 \in A : (a_1 \sqsubseteq_A a_1 \triangledown a_2) \wedge (a_2 \sqsubseteq_A a_1 \triangledown a_2)$;
2. for all ascending sequences $a_0 \sqsubseteq_A \cdots \sqsubseteq_A a_{i+1} \sqsubseteq_A \cdots$, the ascending sequence $x_0 \sqsubseteq_A \cdots \sqsubseteq_A x_{i+1} \sqsubseteq_A \cdots$ defined by $x_0 = a_0$ and $x_{i+1} = x_i \triangledown a_{i+1}$ is not strictly increasing.

The abstract increasing sequence with widening is $x_0 \sqsubseteq_A \cdots \sqsubseteq_A x_{i+1} \sqsubseteq_A \cdots$, where $x_0 \stackrel{\text{def}}{=} \bot_A$ and, for each $i \in \mathbb{N}$,

$$x_{i+1} \stackrel{\text{def}}{=} x_i \triangledown f_A(x_i). \tag{1}$$

Note that condition 1 in Definition 1 states that the widening is an *upper bound operator* for the abstract domain; this makes sure that the computed abstract sequence is indeed increasing, thereby enabling the application of condition 2 and enforcing convergence after a finite number $k \in \mathbb{N}$ of iterations, obtaining $\mathrm{lfp} f_C \sqsubseteq_C \gamma(x_k)$. Roughly speaking, since the classical widening is an upper bound operator, the AI engine can directly use it in Eqn. (1) as a *replacement* of the join operator.

It is well known that the approximation computed by widening during the ascending sequence may be rather coarse and it can be improved by coupling it with a corresponding decreasing sequence using *narrowing operators* [17, 19, 20]; it is also possible to intertwine the ascending and descending phases [4]. Other techniques (e.g., widening up-to [32], lookahead widening [27], stratified widening [43]) are meant to directly improve the precision of the ascending sequence. All of these will not be discussed further, as they are completely orthogonal to the goal of this paper.

3 On the Specification of Widening Operators

In the previous section we have recalled the *classical* specification for the widening operator. However, a quick review of the literature on Abstract Interpretation shows that such a formalization is not uniformly adopted. Quoting from [20, footnote 6]:

> Numerous variants are possible. For example, [. . .]

An incomplete list of the possible variants includes:

- using a different widening for each iterate [14];
- using an n-ary or set-based widening [13, 19], depending on many (possibly all) previous iterates;
- using a widening satisfying the minimalistic specification proposed in [42];
- using a widening for the case when the computational and approximation orderings are different [19, 49].

Some of these variants may be regarded has having (only) theoretical interest, as their application in practical contexts has been quite limited. Here below we describe in more detail the very first variant mentioned in [20, footnote 6], which has been quite successful from a practical point of view, being adopted by a relatively high number of open-source abstract domain libraries and AI tools.

Definition 2 (Alternative widening)

A widening $\triangledown\colon A \times A \to A$ is an operator such that:

1. $\forall a_1, a_2 \in A : a_1 \sqsubseteq_A a_2 \implies a_2 \sqsubseteq_A a_1 \triangledown a_2$;
2. for all ascending sequences $a_0 \sqsubseteq_A \cdots \sqsubseteq_A a_{i+1} \sqsubseteq_A \cdots$, the ascending sequence $x_0 \sqsubseteq_A \cdots \sqsubseteq_A x_{i+1} \sqsubseteq_A \cdots$ defined by $x_0 = a_0$ and $x_{i+1} = x_i \triangledown a_{i+1}$ is not strictly increasing.

When this alternative specification for the widening is adopted, then the (alternative) abstract increasing sequence with widening $x_0 \sqsubseteq_A \cdots \sqsubseteq_A x_{i+1} \sqsubseteq_A \cdots$ is defined by $x_0 \stackrel{\text{def}}{=} \bot_A$ and, for each $i \in \mathbb{N}$,

$$x_{i+1} \stackrel{\text{def}}{=} x_i \triangledown \big(x_i \sqcup_A f_A(x_i)\big). \tag{2}$$

When comparing the alternative and classical specifications, we see that:

- in condition 1 of Definition 2, it is *required* that $a_1 \sqsubseteq_A a_2$, so that this widening is an upper bound operator only when this precondition is satisfied;
- in order for this assumption to hold, in Eqn. (2) the AI engine uses the join $x_i \sqcup_A f_A(x_i)$ as the second argument for the widening.

In other words, when adopting this alternative specification, the widening is meant to be used *in addition to* the join operator, rather than as a *replacement* of the join as was the case for the classical specification.

The option of choosing between the classical and alternative widening specifications has also been recalled more recently in [15]: in the main body of that paper it is assumed that the abstract semantic function f_A is extensive, thereby fulfilling the precondition of Definition 2; in [15, footnote 5] it is observed that the extensivity requirement can be satisfied by using Eqn. (2); the classical widening specification is instead described in [15, footnote 6].

When considering parametric AI-based tools, the minor differences highlighted in the comparison above are nonetheless very important from a practical point of view, since they are directly affecting the interface boundary between the AI engine component and the abstract domain component, which are quite often developed independently. It is therefore essential to classify these components depending on the widening specification they are adopting. In the following, we classify a subset of the available abstract domains and AI tools: we will focus on numerical abstract domains, but the reasoning can be extended to other abstract domains (e.g., the domain of finite state automata [22] used for the analysis of string values [5]); also, we only consider open-source implementations, since our classification is merely based on source code inspection.

3.1 Classifying abstract domain implementations

The classification of an abstract domain *implementation* is based on checking whether the corresponding widening operator is modeled according to Definition 1 or 2, i.e., whether or not it requires the precondition $a_1 \sqsubseteq_A a_2$. We consider most of the domains provided by the open-source libraries APRON (Analyse de PROgrammes Numériques) [37], ELINA (ETH LIbrary for Numerical Analysis) [23], PPL (Parma Polyhedra Library) [7], PPLite [8] and VPL (Verified Polyhedron Library) [11], as well as a few abstract domains that are embedded in specific AI tools, such as Frama-C [38], IKOS [12] and Jandom [1]. The results of our classification are summarized in Table 1.

abstract domain	*classical widening*: Def. 1 (no precondition)	**alternative widening**: Def. 2 (requires $a_1 \sqsubseteq_A a_2$)
intervals/boxes [16, 17]	APRON*, IKOS, Jandom	PPL
zones [40]	ELINA*, IKOS	PPL
octagons [40]	APRON*, ELINA*, IKOS, Jandom	PPL
polyhedra [21, 30]		APRON, ELINA, PPL, PPLite, VPL
parallelotopes [3]	Jandom	
interval congruences [39]		Frama-C

Table 1 Classifying implementations of abstract domains based on widening specification

We discuss in some detail the case of the domain of *intervals* (which generalizes to multi-dimensional boxes).

Definition 3 (Interval abstract domain)
The lattice of intervals on $\mathbb{I} \in \{\mathbb{Z}, \mathbb{Q}\}$ has carrier

$$\mathrm{Itv} = \{\bot\} \cup \left\{ [\ell, u] \;\middle|\; \ell \in \mathbb{I} \cup \{-\infty\}, u \in \mathbb{I} \cup \{+\infty\}, \ell \leq u \right\},$$

partially ordered by $\bot \sqsubseteq x$, for all $x \in \text{Itv}$, and

$$[\ell_1, u_1] \sqsubseteq [\ell_2, u_2] \iff \ell_1 \geq \ell_2 \wedge u_1 \leq u_2.$$

Almost all implementations of this domain adopt the definition of interval widening provided in [16, 17], matching the classical specification of Definition 1:

Definition 4 (Classical widening on intervals)

The interval widening $\triangledown : \text{Itv} \times \text{Itv} \to \text{Itv}$ is defined by $x \triangledown \bot \stackrel{\text{def}}{=} \bot \triangledown x \stackrel{\text{def}}{=} x$, for all $x \in \text{Itv}$, and $[\ell_1, u_1] \triangledown [\ell_2, u_2] \stackrel{\text{def}}{=} [(\ell_2 < \ell_1 \ \mathbf{?} -\infty \ \mathbf{:} \ \ell_1), (u_2 > u_1 \ \mathbf{?} +\infty \ \mathbf{:} \ u_1)]$.

In contrast, the implementation provided by the PPL [7] is based on the alternative widening specification of Definition 2:

Definition 5 (Alternative widening on intervals)

The interval widening $\triangledown : \text{Itv} \times \text{Itv} \to \text{Itv}$ is defined by $\bot \triangledown x \stackrel{\text{def}}{=} x$, for all $x \in \text{Itv}$, and $[\ell_1, u_1] \triangledown [\ell_2, u_2] \stackrel{\text{def}}{=} [(\ell_2 \neq \ell_1 \ \mathbf{?} -\infty \ \mathbf{:} \ \ell_1), (u_2 \neq u_1 \ \mathbf{?} +\infty \ \mathbf{:} \ u_1)]$.

One may wonder what Definition 5 is going to gain with respect to Definition 4: technically speaking, the main motivation for adding a precondition to a procedure is to enable optimizations; for instance, when considering intervals with arbitrary precision rational bounds, testing $r_1 \neq r_2$ is more efficient than testing $r_1 < r_2$.[1]

Weakly relational domains, such as *zones* and *octagons* [40], are characterized by widening operators that are almost identical to those defined on intervals (these three domains can be seen as instances of the template polyhedra construction [46]). We can thus repeat the observations made above to conclude that almost all implementations adopt the classical widening specification, while the PPL implementation keeps adopting the alternative one.

The case of library APRON deserves a rather technical, pedantic note. All the abstract domains provided by this library are meant to implement a common interface, whose C language documentation states the following:[2]

```
ap_abstract1_t ap_abstract1_widening(ap_manager_t* man,
                                     ap_abstract1_t* a1,
                                     ap_abstract1_t* a2)
Widening of a1 with a2. a1 is supposed to be included in a2.
```

That is, all the domains in APRON, including boxes and octagons, are meant to implement the alternative widening specification; hence, even though the actual source code implementing the widenings for boxes has always been based on Definition 4 (up to version 0.9.13), in principle any future release of the library may decide to change its inner implementation details and switch to Definition 5. However, since this is an unlikely scenario, in Table 1 we keep these implementations in the column of the classical widening (marking them with a star). A similar observation applies

[1] Arbitrary precision rationals are usually represented as canonicalized fractions; hence, testing for equality is relatively cheap, whereas testing $\frac{n_1}{d_1} < \frac{n_2}{d_2}$ may require two arbitrary precision products.

[2] Similar sentences are found in the documentation of the C++ and OCaml interfaces.

to the domains in the ELINA library since, as far as we can tell, its programming interface is meant to be identical to that of APRON; in this case, however, there is no clear statement in the documentation.

Things are rather different when considering the domain of *convex polyhedra*: even though the standard widening operator defined in [21, 30] is modeled according to the classical specification, it turns out that *all* the available implementations are adopting the alternative specification of Definition 2 (no matter if considering the standard widening of [21, 30] or the more precise widenings defined in [6, 8]).

The expert reader might note that Table 1 provides a rather incomplete summary, lacking entries for many numerical domains. We stress that the only purpose of this table is to witness a non-uniform situation; this is further confirmed by the last two lines of the table, where we have considered a couple of less popular abstract domain implementations (*parallelotopes* [3] and *interval congruences* [39]).

It is also instructive to extend our review to consider the case of an abstract domain satisfying the ACC (Ascending Chain Condition), meaning that all its ascending chains are finite. We still require a widening operator for the following reasons:

1. to obtain a uniform abstract domain interface, which simplifies the generic implementation of AI engines;
2. to accelerate (rather than ensure) the convergence of the abstract iteration sequence, when the abstract domain contains very long increasing chains.

When the last reason above does not apply, an obvious option is to define a *dummy* widening operator, intuitively based on the abstract join operator $\sqcup_A$, so as to avoid precision losses as much as possible. Once again, we are faced with the choice of adopting the classical specification of Definition 1, leading to

$$a_1 \triangledown a_2 \stackrel{\text{def}}{=} a_1 \sqcup_A a_2, \tag{3}$$

or the alternative specification of Definition 2, exploiting precondition $a_1 \sqsubseteq_A a_2$ to improve efficiency, and hence obtaining

$$a_1 \triangledown a_2 \stackrel{\text{def}}{=} a_2. \tag{4}$$

As an example, the *sign domain* [17] found in the Frama-C framework implements its dummy widening according to Eqn. (3):

```
sign_domain.ml: let widen = join
```

whereas the same framework implements the widening for the *generic lattice set domain* according to Eqn. (4):[3]

```
abstract_interp.ml: let widen _wh _t1 t2 = t2
```

This is somehow surprising, since one would expect that all the abstract domains implemented in a given framework are developed according to the same specification.

[3] Parameter `_wh`, which is ignored, is meant to provide *widening hints* [10].

3.2 Classifying AI engine implementations

We now turn our attention to the other side of the interface, checking how widening operators are used in different AI engines, that is whether they use widenings to replace joins, as in Eqn. (1), or in addition to joins, as in Eqn. (2). The results of this second classification are summarized in Table 2. As before, we are focusing on a rather limited subset of the available open-source AI engines, since our only goal is to show that, again, we have a non-uniform situation.

classical engine: Eqn. (1) (only widening)	*alternative engine*: Eqn. (2) (join + widening)
CPAchecker [9]	(CPAchecker + APRON polyhedra)
DIZY [45]	Frama-C [38]
GoLiSA [44]	Goblint [50]
IKOS [12]	Interproc [36]
Jandom [1]	(Jandom + PPL domains)
LiSA [24]	PAGAI [34]
SeaHorn/Crab [29]	(SeaHorn/Crab + Boxes)

Table 2 Classifying AI engines based on widening specification

It is worth stressing that, in this case, the classification process is more difficult, because there are AI tools that try to avoid the widening specification mismatch by adapting their own AI engine depending on the abstract domain chosen for the analysis. For instance, CPAchecker, Jandom and SeaHorn/Crab are all classified to follow the classical specification, as they usually only apply the widening operator. However, they switch to the alternative specification (computing joins before widenings) when using, respectively, the polyhedra domain provided by APRON, the abstract domains provided by PPL and the abstract domain of Boxes [28]; these special cases are reported in Table 2 in parentheses.[4]

4 Combinations of Abstract Domains and AI Engines

In the previous section we have seen that widening implementation in abstract domains and widening usage in AI engines can be classified according to the specification they are meant to follow. Here we consider all possible ways to combine the two components inside an AI tool. Even though we will often refer to combinations that are actually found in some of the available tools, the discussion is meant to be more general: that is, we also target potential combinations of engines and domains

[4] Our classification is based on human code review, hence it is error prone: we may have missed other specific combinations treated as special cases.

that have never been implemented in the considered AI tools; the idea is that these combinations are anyway possible when assuming a collaborative context.[5]

The possible combinations are classified in Table 3, where we highlight in *italic blue* the *safe* portions of the specification (i.e., the parts making sure that we will obtain a correct analysis result), while highlighting in **boldface red** the **risky** portions of the specification (whose correctness depends on assumptions that should be satisfied by the *other* interfaced component). Also note that in the table, for exposition purposes, we use **IKOS** (resp., *PAGAI*) to actually represent *any* AI engine adopting the **classical** (resp., *alternative*) engine specification; similarly, we use APRON's *octagons* (resp., **polyhedra**) to represent *any* abstract domain whose widening is implemented according to the *classical* (resp., **alternative**) widening specification.

AI engine \ widening	*classical*: Def. 1 (no precondition)	**alternative**: Def. 2 (requires $a_1 \sqsubseteq_A a_2$)
classical: Eqn. (1) (only widening)	**IKOS** + *octagons*	**IKOS** + **polyhedra**
alternative: Eqn. (2) (join + widening)	*PAGAI* + *octagons*	*PAGAI* + **polyhedra**

Table 3 Classifying possible combinations of AI engines and widenings

We now briefly describe the four kinds of combination.

IKOS + *octagons*. This combination matches the classical specification in [16, 17, 20], whose safety is ensured by Definition 1.

PAGAI + **polyhedra**. This combination matches the alternative widening specification of [20, footnote 6], whose safety is ensured by Eqn. (2). The combination *Frama-C* + **generic lattice set** mentioned previously can be seen to be another instance of this kind, where the abstract domain satisfies the ACC.

PAGAI + *octagons*. This combination, which mixes the classical and alternative specifications, is probably not explicitly described in the literature; it can thus be interpreted as the result of a harmless specification mismatch. The safety of the resulting analysis is clearly ensured (by adopting a *belt and suspenders* approach). The combination *Frama-C* + *signs* mentioned previously can be seen to be another instance of this hybrid approach, where the abstract domain satisfies the ACC.

IKOS + **polyhedra**. This combination is once again mixing the classical and alternative specifications, but in this case *both* the AI engine and the widening implementation take a risky approach, assuming that the other component will do what is required to obtain safety. Since neither assumption is satisfied, safety is (potentially) compromised.

[5] Clearly, such an ideal scenario is sometimes limited by licencing issues and/or architectural and programming language implementation barriers.

In summary, of the four possible combination kinds, the first three are safe and the last one is (potentially) unsafe.

4.1 Some thoughts on the unsafe combinations

Technically speaking, we are witnessing the violation of a programming contract: depending on the implementation details, the outcome could be either the raising of a well-defined error or an *undefined behavior*. In the first (unlikely) case, the implementers are left with two options: either they simply report the problem to the user, interrupting the static analysis process, or they workaround the issue by arbitrarily returning a safe approximation of the actual result (e.g., the uninformative top element $\top_A \in A$). In the more likely case of undefined behavior, the outcome is obviously unpredictable. For instance, a few experiments on the unsafe combination **PAGAI** + **PPLite polyhedra**[6] resulted in observing any of the following:

- identical results;
- different safe results (which usually were precision losses);
- unsafe results;
- non termination of the analysis;
- segmentation faults.

It is worth stressing that this issue is well known by AI experts. For instance, in the case of convex polyhedra, a full blown example showing the risk of missing the inclusion requirement was provided in [6, Example 5]; the problem has also been recalled more recently in [31]:

> [. . .] widening operators are generally designed under the assumption that their first operand is smaller than the second one.

As we already observed, the precondition $a_1 \sqsubseteq_A a_2$ is often mentioned in the documentation of software libraries (e.g., APRON, PPL and VPL) and in some of them the implementers have added assertions to check it; however, these assertions are executed only in debugging mode, which is often avoided for efficiency reasons.

The case of VPL [11] deserves a longer note. As its name suggests, the Verified Polyhedron Library is meant to support *verified* computations on the domain of convex polyhedra: that is, each library operation, besides providing the result, also yields a *correctness certificate*; the result and the certificate can then be supplied to a verification procedure, which is formally proved correct in Coq. Therefore, the widening specification mismatch under investigation would have been a *perfect fit* for the VPL setting. Unfortunately, the library developers decided that widening is the one and only operator on polyhedra *not* requiring a certificate. Quoting from VPL's documentation:

```
Note that [widen p1 p2] relies on [p1] being included in [p2].
The result includes the two operands, although no certificate
is created.
```

[6] As shown in Table 2, *PAGAI* is a safe AI engine; **PAGAI** is a variant obtained by removing the computation of joins before widenings, i.e., replacing on purpose Eqn. (2) with Eqn. (1).

Hence, VPL cannot promptly detect this kind of errors. In [25] it is argued that widening operators need not be certified because the framework is anyway going to formally check the computed post-fixpoint for inductiveness, so that any unsafe result will be anyway identified (later on). However, as discussed above, unsafety is just one of the possible undesired behaviors.

To summarize, under this non-uniform state of things, it is rather difficult to guarantee that AI engines and abstract domains possibly developed by different research groups are always combined safely. Indeed, a partial review of the literature shows that a few unsafe combinations have been used, in recent years, in at least four artifacts accepted in top level conferences:

1. a SAS 2013 artifact [45], combining **DIZY** with APRON **polyhedra**;
2. a POPL 2017 artifact [47], combining **SeaHorn/Crab** with (APRON, ELINA and PPL) **polyhedra**;
3. a CAV 2018 artifact [48], combining **SeaHorn/Crab** with ELINA **polyhedra**;
4. a PLDI 2020 artifact [33], combining **SeaHorn/Crab** with ELINA **polyhedra**.

The unsafety of the last three artifacts, which are based on the SeaHorn/Crab engine, has been confirmed by one of their authors.[7]

At this point, it might seem natural to question whether or not these AI tools are *actually* computing unsafe results. Note that the answer is far from being obvious, because these analyses might be often returning a correct result anyway: namely, they might be losing safety in one iteration of the abstract sequence just to regain it at a later iteration, possibly due to the usual over-approximations incurred by the abstract semantic operators; this effect might even be amplified by the non-monotonicity of widening operators. We actually conjecture that all these experimental evaluations are almost never affected by the potential safety issue, so that they keep playing their role in the context of the considered papers.

More importantly, it is our opinion that answering the question above is not really interesting: as soon as a potential problem is identified, one should focus on finding an adequate solution for the future, rather than looking for a specific example witnessing an actual error in the past. This will be the goal of Section 5.

4.2 Comparing the safe combinations

We now compare the safe combinations, trying to understand whether there indeed are good reasons to have three slightly different options.

We start by observing that any combination such as *PAGAI* + *octagons* is going to lose some efficiency because the alternative AI engine, by following Eqn. (2), is going to compute joins that are completely useless. Hence, we are left with a comparison of the two combinations corresponding to the classical (**IKOS** + *octagons*) and alternative (*PAGAI* + **polyhedra**) specifications.

[7] G. Singh, personal communication.

Firstly, we should get rid of a *false impression* related to the efficiency of widenings. When adopting the point of view of the abstract domain implementer, one may think that the alternative specification is going to be less costly because it makes possible to exploit the inclusion precondition without having to compute the expensive join (recall the observation we made just after Definition 5). However, when considering the AI tool as a whole, we can easily note that the computation of the join is not really avoided: rather, it is just delegated to the AI engine component, so that any corresponding overhead is still there.

Secondly, we will argue that the classical widening specification, by *merging* join and widening into a single operator, can sometimes trigger improvements in precision and/or efficiency. A potential precision improvement can be obtained when considering abstract domains that are not semi-lattices [26], such as the domain of *parallelotopes* [2, 3]. These domains are not provided with a *least* upper bound operator; rather, when computing a join, they necessarily have to choose among alternative, uncomparable upper bound operators: a poor choice typically leads to precision losses. Hence, as observed in [2, 3], forcing a separation between the join and the widening (as required in the alternative widening specification) is going to complicate the choice of an appropriate upper bound; in contrast, an implementation based on the classical widening will be able to exploit more context information and preserve precision. By following the same reasoning, one could argue that an implementation based on the classical widening specification may be able to apply specific efficiency optimizations that would be prevented when separating the join from the widening; we will show a concrete example in this respect in Section 5.1.

To summarize, among the safe combinations, those based on the classical widening specification seem to have a few advantages and no disadvantage at all.

5 Lesson learned and recommendation

The discussion in the previous section has shown that having different, not fully compatible specifications for the widening operators implemented in abstract domains and used by AI engines can cause issues that potentially affect the overall correctness of the analysis. Knowledgeable software engineers implementing AI tools may be easily confused and even AI experts may fail to detect these specification mismatches, since their consequences are quite often hidden.

In our humble opinion, this state of things should be corrected: when implementing parametric static analysis tools following the traditional AI approach, we should "fix" (i.e., stick to) a single and universally adopted specification for the widening operators. For all we said in the previous section, our recommendation is to choose the classical widening of [16, 17]. Namely,

- the abstract domain designers should follow Definition 1 and make sure that their widening is an upper bound operator (with no precondition), thereby solving all the correctness issues; and

- the AI engine designers should always adopt Eqn. (1), which slightly improves efficiency by avoiding useless joins and can also preserve precision when the analysis is using a non-lattice abstract domain.

It is worth stressing that, from a technical point of view, following the two recommendations above is quite simple: there is no new functionality that needs to be implemented; rather, the AI engine developers and the abstract domain implementers only need to agree on a clearer separation of concerns. For instance, an abstract domain implementing the alternative widening specification will have to *wrap* its implementation using an upper bound operator, as follows.

Proposition 1 *Let* $\nabla_{\textit{risky}} \colon A \times A \to A$ *be a widening operator satisfying Definition 2 and let* $\tilde{\sqcup}_A \colon A \times A \to A$ *be an upper bound operator on* A *(not necessarily the least one); then, the operator* $\nabla_{\textit{safe}} \colon A \times A \to A$ *defined by*

$$\forall a_1, a_2 \in A : a_1 \nabla_{\textit{safe}} a_2 \stackrel{\text{def}}{=} a_1 \nabla_{\textit{risky}} (a_1 \tilde{\sqcup}_A a_2)$$

is a widening satisfying Definition 1.

Note that the widening wrapping operation is often (already) implemented inside classical AI engines that correctly interface an abstract domain implementing the alternative widening specification. Hence, we are just suggesting to *move* this wrapping operation (once and for all) into the abstract domain's widening implementation, thereby simplifying the parametric AI engine. In specific cases, the wrapped widening of Proposition 1 can be replaced by an *ad hoc* implementation meant to improve its precision and/or its efficiency.

5.1 Safe widenings for convex polyhedra

As an interesting example, we discuss in more detail the case of the abstract domain of topologically closed convex polyhedra, describing the several implementation options that allow to transform a **risky** widening implementation (as found in any one of the software libraries we considered) into a *safe* widening. For space reasons, we will provide abridged definitions of the domain and its operators, referring the interested reader to the well known literature [6, 21, 30, 46].

Let $\langle \mathbb{CP}_n, \subseteq, \uplus, \emptyset, \mathbb{R}^n \rangle$ be the semi-lattice of n-dimensional topologically closed convex polyhedra, where the least upper bound operator $\uplus$ denotes the convex polyhedral hull; let also ∇_{std} be an implementation of the standard widening operator assuming the precondition $P_1 \subseteq P_2$. Clearly, the simplest option is to instantiate Proposition 1 using $\uplus$, thereby obtaining

$$P_1 \nabla_{\text{a}} P_2 \stackrel{\text{def}}{=} P_1 \nabla_{\text{std}} (P_1 \uplus P_2). \tag{5}$$

However, as observed in [46], this could result in some waste of computational resources. Roughly speaking, the standard widening $P_1 \nabla_{\text{std}} P_2$ selects the constraints

of P_1 that are satisfied by P_2. Hence, there is no real need to compute the *least* upper bound of the two polyhedra, which is also going to compute new constraint slopes. Rather, we can instantiate Proposition 1 by using the *weak join* operator $\sqcup_{\mathrm{w}}$,[8] which does not compute new constraint slopes:

$$P_1 \ \triangledown_{\mathrm{b}} \ P_2 \stackrel{\text{def}}{=} P_1 \ \triangledown_{\mathrm{std}} \ (P_1 \sqcup_{\mathrm{w}} P_2). \tag{6}$$

A third option is to provide an *ad hoc* definition for the widening on polyhedra, which simply avoids the explicit computation of least/weak upper bounds.

Definition 6 (*Ad hoc* classical widening on $\mathbb{CP}_n$)

Let $P_1, P_2 \in \mathbb{CP}_n$ be represented by the constraint system C_1 and the generator system G_2, respectively, where $C_1 = Eqs_1 \cup Ineqs_1$ is in minimal form. Consider polyhedron P represented by $C \stackrel{\text{def}}{=} C_1 \setminus \{\, c_1 \in Ineqs_1 \mid g_2 \in G_2 \text{ violates } c_1 \,\}$. Then,

$$P_1 \ \triangledown_{\mathrm{c}} \ P_2 \stackrel{\text{def}}{=} \begin{cases} P_1 \uplus P_2, & \text{if } (P_1 = \emptyset) \text{ or } (P_2 = \emptyset) \text{ or } (g_2 \in G_2 \text{ violates } c_1 \in Eqs_1); \\ P, & \text{otherwise.} \end{cases}$$

Note that this is a well defined (i.e., semantic) widening operator, meaning that it does not depend on the constraint representation. This property holds because:

1. it considers a constraint system C_1 in minimal form;
2. it returns $P_1 \uplus P_2$ whenever a generator in G_2 violates an *equation* in C_1.

Hence, the polyhedron P is computed only when P_1 and $P_1 \uplus P_2$ have the same affine dimension (see [6, Proposition 6]). Also note that, in case 2 above, the computed result can be more precise than the one obtained using Eqn. (5) or Eqn. (6).

time/operations	$P_1 \ \triangledown_{\mathrm{a}} \ P_2$	$P_1 \ \triangledown_{\mathrm{b}} \ P_2$	$P_1 \ \triangledown_{\mathrm{c}} \ P_2$	$\triangledown_{\mathrm{a}}/\triangledown_{\mathrm{b}}$	$\triangledown_{\mathrm{a}}/\triangledown_{\mathrm{c}}$
cumulative time	163 ms	287 ms	19 ms	0.57	8.58
scalar products	808078	1153298	131482	0.70	6.15
linear combinations	42527	70162	4456	0.61	9.54
bitset operations	5881982	8556088	106031	0.69	5.55

Table 4 Efficiency comparison for variations of standard widening on $\mathbb{CP}_n$

In Table 4 we summarize the results of an experimental evaluation (using a modified version of the PPLite library) where we compare the efficiency of the operators $\triangledown_{\mathrm{a}}$, $\triangledown_{\mathrm{b}}$ and $\triangledown_{\mathrm{c}}$; note that the use of operator $\triangledown_{\mathrm{a}}$ faithfully describes an implementation satisfying the alternative specification of Definition 2. The reader is warned that we are describing a *synthetic* efficiency comparison, having little statistical value: usually, a rather small fraction of the overall execution time of static analysis tools is spent computing widenings. In our synthetic test, we consider 70 pairs of randomly generated, fully dimensional closed polyhedra in a vector space

[8] This is the join used in the domain of template polyhedra [46], also called *constraint hull*.

of dimension 5 (each polyhedron is obtained by adding 5 random rays to a randomly generated bounded box). For each pair, we compute the widenings according to the three implementations, keeping track of the overall elapsed time; note that the three widenings always compute identical results. We also repeat the test with a modified implementation that tracks the number of low level operations (scalar products, linear combinations and bitwise operations on saturation vectors) executed during the computations of the widenings. Table 4 shows that, on these benchmarks, the *ad hoc* classical widening ∇_{c} is able to significantly improve the efficiency of ∇_{a}. In contrast, for ∇_{b} (which is based on the weak join operator) we obtain a slowdown; this is due to the fact that, after efficiently computing the constraint system for $P_1 \sqcup_{\mathrm{w}} P_2$, the conversion procedure is implicitly invoked to obtain the corresponding generator system (which is required by the current implementation of ∇_{std}).

5.2 A note on the unusual widening specifications

Our recommendation is meant for tools following the mainstream approach to Abstract Interpretation. In particular, we are assuming that the computational ordering is identical to the approximation ordering. The rather unusual case of *distinct* orderings $\sqsubseteq_A$ and $\preceq_A$ has been first considered in [19, Proposition 6.20]: in this case, non-standard hypotheses on the widening operator make sure that the approximation relation $\alpha(c_i) \preceq_A a_i$ holds for each corresponding pair of concrete and abstract iterates (these properties can then be transferred to the fixpoint).

The decision tree abstract domain proposed in [49], which can be used to prove conditional termination of programs, is an interesting specific example of a domain distinguishing the computational and approximation ordering. The corresponding widening operator, however, matches none of the specifications above: neither Definition 1, nor Definition 2, nor [19, Proposition 6.20]. As discussed in [49], when using this widening operator, the approximation relation $\alpha(c_i) \preceq_A a_i$ on the intermediate concrete and abstract iterates does *not* generally hold: the analysis can only ensure the correct over-approximation of the concrete fixpoint.

6 Conclusion

We have shown that widely adopted open-source abstract domain libraries and AI engines assume slightly different specifications for widening operators, possibly leading to AI tool crashes or more subtle, hidden safety issues. This problem can be solved by systematically adopting a *fixed* and hence uniform widening specification. Based on our investigation, the best option is to stick to the *classical* widening specification of [16, 17], as it enjoys the following properties:

- it is the default one taught in Abstract Interpretation courses and tutorials;
- it avoids all safety issues;

- it can be more precise than the alternative one (for non-lattice domains);
- its implementations can be as efficient as (or even more efficient than) those based on the alternative one.

Acknowledgements The authors would like to express their gratitude to the developers and maintainers of open-source abstract domain libraries and static analysis tools. Special thanks are also due to Gianluca Amato, David Monniaux, Jorge A. Navas, Francesca Scozzari, Gagandeep Singh, and Caterina Urban for their helpful feedback on the subject of this paper.

References

1. Amato, G., Di Nardo Di Maio, S., Scozzari, F.: Numerical static analysis with Soot. In: P. Lam, E. Sherman (eds.) Proceedings of the 2nd ACM SIGPLAN International Workshop on State Of the Art in Java Program analysis, SOAP 2013, Seattle, WA, USA, June 20, 2013, pp. 25–30. ACM (2013). DOI 10.1145/2487568.2487571
2. Amato, G., Rubino, M., Scozzari, F.: Inferring linear invariants with parallelotopes. Sci. Comput. Program. **148**, 161–188 (2017). DOI 10.1016/j.scico.2017.05.011
3. Amato, G., Scozzari, F.: The abstract domain of parallelotopes. Electron. Notes Theor. Comput. Sci. **287**, 17–28 (2012). DOI 10.1016/j.entcs.2012.09.003
4. Amato, G., Scozzari, F., Seidl, H., Apinis, K., Vojdani, V.: Efficiently intertwining widening and narrowing. Sci. Comput. Program. **120**, 1–24 (2016). DOI 10.1016/j.scico.2015.12.005
5. Arceri, V., Mastroeni, I.: Analyzing dynamic code: A sound abstract interpreter for *Evil* eval. ACM Trans. Priv. Secur. **24**(2), 10:1–10:38 (2021). DOI 10.1145/3426470
6. Bagnara, R., Hill, P., Ricci, E., Zaffanella, E.: Precise widening operators for convex polyhedra. Sci. Comput. Program. **58**(1-2), 28–56 (2005). DOI 10.1016/j.scico.2005.02.003
7. Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Sci. Comput. Program. **72**(1-2), 3–21 (2008). DOI 10.1016/j.scico.2007.08.001
8. Becchi, A., Zaffanella, E.: PPLite: zero-overhead encoding of NNC polyhedra. Inf. Comput. **275**, 104620 (2020). DOI 10.1016/j.ic.2020.104620
9. Beyer, D., Keremoglu, M.E.: CPAchecker: A tool for configurable software verification. In: G. Gopalakrishnan, S. Qadeer (eds.) Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings, *Lecture Notes in Computer Science*, vol. 6806, pp. 184–190. Springer (2011). DOI 10.1007/978-3-642-22110-1_16
10. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. In: R. Cytron, R. Gupta (eds.) Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation 2003, San Diego, California, USA, June 9-11, 2003, pp. 196–207. ACM (2003). DOI 10.1145/781131.781153
11. Boulmé, S., Maréchal, A., Monniaux, D., Périn, M., Yu, H.: The Verified Polyhedron Library: an overview. In: 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2018, Timisoara, Romania, September 20-23, 2018, pp. 9–17. IEEE (2018). DOI 10.1109/SYNASC.2018.00014
12. Brat, G., Navas, J., Shi, N., Venet, A.: IKOS: A framework for static analysis based on abstract interpretation. In: D. Giannakopoulou, G. Salaün (eds.) Software Engineering and Formal Methods - 12th International Conference, SEFM 2014, Grenoble, France, September 1-5, 2014. Proceedings, *Lecture Notes in Computer Science*, vol. 8702, pp. 271–277. Springer (2014). DOI 10.1007/978-3-319-10431-7_20
13. Cortesi, A., Zanioli, M.: Widening and narrowing operators for abstract interpretation. Comput. Lang. Syst. Struct. **37**(1), 24–42 (2011). DOI 10.1016/j.cl.2010.09.001

14. Cousot, P.: Semantic foundations of program analysis. In: S.S. Muchnick, N.D. Jones (eds.) Program Flow Analysis: Theory and Applications, chap. 10, pp. 303–342. Prentice Hall, Englewood Cliffs, NJ, USA (1981)
15. Cousot, P.: Abstracting induction by extrapolation and interpolation. In: D. D'Souza, A. Lal, K.G. Larsen (eds.) Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings, *Lecture Notes in Computer Science*, vol. 8931, pp. 19–42. Springer (2015). DOI 10.1007/978-3-662-46081-8_2
16. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: Proceedings of the Second International Symposium on Programming, pp. 106–130. Dunod, Paris, France (1976)
17. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977, pp. 238–252 (1977)
18. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, Texas, USA, January 1979, pp. 269–282 (1979)
19. Cousot, P., Cousot, R.: Abstract interpretation frameworks. J. Log. Comput. **2**(4), 511–547 (1992). DOI 10.1093/logcom/2.4.511
20. Cousot, P., Cousot, R.: Comparing the galois connection and widening/narrowing approaches to abstract interpretation. In: M. Bruynooghe, M. Wirsing (eds.) Programming Language Implementation and Logic Programming, 4th International Symposium, PLILP'92, Leuven, Belgium, August 26-28, 1992, Proceedings, *Lecture Notes in Computer Science*, vol. 631, pp. 269–295. Springer (1992). DOI 10.1007/3-540-55844-6_142
21. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: A. Aho, S. Zilles, T. Szymanski (eds.) Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978, pp. 84–96. ACM Press (1978). DOI 10.1145/512760.512770
22. D'silva, V.: Widening for automata. Diploma thesis, Institut Fur Informatick, Universitat Zurich, Switzerland (2006)
23. ETH Zurich SRI Lab: ELINA: ETH Library for Numerical Analysis. URL http://elina.ethz.ch
24. Ferrara, P., Negrini, L., Arceri, V., Cortesi, A.: Static analysis for dummies: experiencing LiSA. In: L. Nguyen Quang Do, C. Urban (eds.) SOAP@PLDI 2021: Proceedings of the 10th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis, Virtual Event, Canada, 22 June, 2021, pp. 1–6. ACM (2021). DOI 10.1145/3460946.3464316
25. Fouilhé, A.: Revisiting the abstract domain of polyhedra : constraints-only representation and formal proof. (le domaine abstrait des polyèdres revisité : représentation par contraintes et preuve formelle). Ph.D. thesis, Grenoble Alpes University, France (2015)
26. Gange, G., Navas, J.A., Schachte, P., Søndergaard, H., Stuckey, P.J.: Abstract interpretation over non-lattice abstract domains. In: F. Logozzo, M. Fähndrich (eds.) Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings, *Lecture Notes in Computer Science*, vol. 7935, pp. 6–24. Springer (2013). DOI 10.1007/978-3-642-38856-9_3
27. Gopan, D., Reps, T.: Lookahead widening. In: T. Ball, R. Jones (eds.) Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings, *Lecture Notes in Computer Science*, vol. 4144, pp. 452–466. Springer (2006). DOI 10.1007/11817963_41
28. Gurfinkel, A., Chaki, S.: Boxes: A symbolic abstract domain of boxes. In: R. Cousot, M. Martel (eds.) Static Analysis - 17th International Symposium, SAS 2010, Perpignan, France, September 14-16, 2010. Proceedings, *Lecture Notes in Computer Science*, vol. 6337, pp. 287–303. Springer (2010). DOI 10.1007/978-3-642-15769-1_18
29. Gurfinkel, A., Navas, J.A.: Abstract interpretation of LLVM with a region-based memory model. In: R. Bloem, R. Dimitrova, C. Fan, N. Sharygina (eds.) Software Verification - 13th

International Conference, VSTTE 2021, New Haven, CT, USA, October 18-19, 2021, and 14th International Workshop, NSV 2021, Los Angeles, CA, USA, July 18-19, 2021, Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 13124, pp. 122–144. Springer (2021). DOI 10.1007/978-3-030-95561-8_8

30. Halbwachs, N.: Détermination automatique de relations linéaires vérifiées par les variables d'un programme. Ph.D. thesis, Grenoble Institute of Technology, France (1979)
31. Halbwachs, N., Henry, J.: When the decreasing sequence fails. In: A. Miné, D. Schmidt (eds.) Static Analysis - 19th International Symposium, SAS 2012, Deauville, France, September 11-13, 2012. Proceedings, *Lecture Notes in Computer Science*, vol. 7460, pp. 198–213. Springer (2012). DOI 10.1007/978-3-642-33125-1_15
32. Halbwachs, N., Proy, Y., Raymond, P.: Verification of linear hybrid systems by means of convex approximations. In: B.L. Charlier (ed.) Static Analysis, First International Static Analysis Symposium, SAS'94, Namur, Belgium, September 28-30, 1994, Proceedings, *Lecture Notes in Computer Science*, vol. 864, pp. 223–237. Springer (1994). DOI 10.1007/3-540-58485-4_43
33. He, J., Singh, G., Püschel, M., Vechev, M.T.: Learning fast and precise numerical analysis. In: A.F. Donaldson, E. Torlak (eds.) Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020, pp. 1112–1127. ACM (2020). DOI 10.1145/3385412.3386016
34. Henry, J., Monniaux, D., Moy, M.: PAGAI: A path sensitive static analyser. Electron. Notes Theor. Comput. Sci. **289**, 15–25 (2012). DOI 10.1016/j.entcs.2012.11.003
35. Jeannet, B.: Some experience on the software engineering of abstract interpretation tools. Electron. Notes Theor. Comput. Sci. **267**(2), 29–42 (2010). DOI 10.1016/j.entcs.2010.09.016
36. Jeannet, B., Argoud, M., Lalire, G.: The INTERPROC interprocedural analyzer. URL http://pop-art.inrialpes.fr/interproc/interprocweb.cgi
37. Jeannet, B., Miné, A.: Apron: A library of numerical abstract domains for static analysis. In: A. Bouajjani, O. Maler (eds.) Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings, *Lecture Notes in Computer Science*, vol. 5643, pp. 661–667. Springer (2009). DOI 10.1007/978-3-642-02658-4_52
38. Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-C: A software analysis perspective. Formal Aspects Comput. **27**(3), 573–609 (2015). DOI 10.1007/s00165-014-0326-7
39. Masdupuy, F.: Semantic analysis of interval congruences. In: D. Bjørner, M. Broy, I.V. Pottosin (eds.) Formal Methods in Programming and Their Applications, International Conference, Akademgorodok, Novosibirsk, Russia, June 28 - July 2, 1993, Proceedings, *Lecture Notes in Computer Science*, vol. 735, pp. 142–155. Springer (1993). DOI 10.1007/BFb0039705
40. Miné, A.: Weakly relational numerical abstract domains. Ph.D. thesis, École Polytechnique, Palaiseau, France (2004)
41. Miné, A.: Tutorial on static inference of numeric invariants by abstract interpretation. Found. Trends Program. Lang. **4**(3-4), 120–372 (2017). DOI 10.1561/2500000034
42. Monniaux, D.: A minimalistic look at widening operators. High. Order Symb. Comput. **22**(2), 145–154 (2009). DOI 10.1007/s10990-009-9046-8
43. Monniaux, D., Guen, J.L.: Stratified static analysis based on variable dependencies. Electron. Notes Theor. Comput. Sci. **288**, 61–74 (2012). DOI 10.1016/j.entcs.2012.10.008
44. Olivieri, L., Tagliaferro, F., Arceri, V., Ruaro, M., Negrini, L., Cortesi, A., Ferrara, P., Spoto, F., Talin, E.: Ensuring determinism in blockchain software with GoLiSA: an industrial experience report. In: L. Gonnord, L. Titolo (eds.) SOAP '22: 11th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis, San Diego, CA, USA, 14 June 2022, pp. 23–29. ACM (2022). DOI 10.1145/3520313.3534658
45. Partush, N., Yahav, E.: Abstract semantic differencing for numerical programs. In: F. Logozzo, M. Fähndrich (eds.) Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings, *Lecture Notes in Computer Science*, vol. 7935, pp. 238–258. Springer (2013). DOI 10.1007/978-3-642-38856-9_14
46. Sankaranarayanan, S., Colón, M., Sipma, H.B., Manna, Z.: Efficient strongly relational polyhedral analysis. In: E.A. Emerson, K.S. Namjoshi (eds.) Verification, Model Checking, and

Abstract Interpretation, 7th International Conference, VMCAI 2006, Charleston, SC, USA, January 8-10, 2006, Proceedings, *Lecture Notes in Computer Science*, vol. 3855, pp. 111–125. Springer (2006). DOI 10.1007/11609773_8

47. Singh, G., Püschel, M., Vechev, M.: Fast polyhedra abstract domain. In: G. Castagna, A. Gordon (eds.) Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017, pp. 46–59. ACM (2017). DOI 10.1145/3009837.3009885
48. Singh, G., Püschel, M., Vechev, M.T.: Fast numerical program analysis with reinforcement learning. In: H. Chockler, G. Weissenbacher (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I, *Lecture Notes in Computer Science*, vol. 10981, pp. 211–229. Springer (2018). DOI 10.1007/978-3-319-96145-3_12
49. Urban, C., Miné, A.: A decision tree abstract domain for proving conditional termination. In: M. Müller-Olm, H. Seidl (eds.) Static Analysis - 21st International Symposium, SAS 2014, Munich, Germany, September 11-13, 2014. Proceedings, *Lecture Notes in Computer Science*, vol. 8723, pp. 302–318. Springer (2014). DOI 10.1007/978-3-319-10936-7_19
50. Vojdani, V., Apinis, K., Rõtov, V., Seidl, H., Vene, V., Vogler, R.: Static race detection for device drivers: the Goblint approach. In: D. Lo, S. Apel, S. Khurshid (eds.) Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016, pp. 391–402. ACM (2016). DOI 10.1145/2970276.2970337